Identification /5

Prénom: .....

Nom: .....

Code permanent: .....

Veuillez choisir **une seule réponse** par question, avec une croix (×) ou un crochet (✓).

Q1. Une caractéristique souhaitable pour qu'une application informatique soit modulaire est la facilité de remplacer une composante de l'application par une autre. Quelle est cette caractéristique?

- ■ Interchangeabilité
- ☐ Séparation
- ☐ Réutilisation
- ☐ Spécificité

Q2. Quel est le type fourni par la bibliothèque `stdarg.h` permettant de définir des fonctions variadiques?

- ☐ `va_arg`
- ■ `va_list`
- ☐ `va_end`
- ☐ `va_start`

Q3. Qu'est-ce qui sera affiché sur la sortie standard par le programme suivant?

```
#include <stdio.h>

#define abs(X) ((X) >= 0 ? X : -X)
#define dist(X, Y) abs(X - Y)

int main(void) {
  int a = 2;
  printf("%d\n", dist(-1, 5 - a));
  return 0;
}
```

- ☐ `-2`
- ☐ `4`
- ■ `-6`
- ☐ `8`

Q4. Considérons un module régulier C composé des fichiers `module.h` et `module.c`. Parmi les éléments suivants, lequel ne devrait pas apparaître dans le fichier `module.c`?

- ☐ Une déclaration de type
- ☐ Une déclaration de constante
- ■ Une protection contre inclusions multiples
- ☐ Une macro-fonction

Q5. Considérez la déclaration suivante, vue en classe, qui permet de représenter une matrice 2D:

```
// Une matrice 2D
struct Matrix {
  unsigned int r;  // Nombre de lignes
  unsigned int c;  // Nombre de colonnes
  double **values; // Valeurs
};
```

Proposez une implémentation de la fonction

```
bool is_symmetric(const struct Matrix* m)
```

qui indique si la matrice `m` est symétrique. Vous pouvez supposer que `m` est une matrice qui a été construite de façon valide. *Rappel:* une matrice est dite symétrique si elle est égale à elle-même lorsqu'on la transpose (on applique une réflexion par rapport à sa diagonale principale). *Note:* utilisez le verso au besoin.

Voir solution page suivante

Q1. La séparation modulaire vise à diviser les préoccupations en composantes. La réutilisation d'un module vise à offrir la possibilité de l'utiliser dans différents contextes. La spécificité d'un module, quant à elle, indique que celui-ci règle une préoccupation précise. C'est donc l'interchangeabilité qui correspond à la description donnée plus haut.

Q2. Le type `va_list` permet de représenter une liste d'arguments d'une fonction variadique. Les identifiants `var_arg`, `var_end` et `va_start` sont des fonctions qui facilitent la manipulation d'instances de type `va_list`.

Q3. Il est important de garder en tête que les macro-fonctions font des substitutions purement textuelles des arguments. En remplaçant successivement les expressions, on obtient ceci:

```
dist(-1, 5 - a)  =  abs(-1 - 5 - a)
                 =  ((-1 - 5 - a) >= 0 ? -1 - 5 - a : --1 - 5 - a)
                 =  ((-1 - 5 - 2) >= 0 ? -1 - 5 - 2 : --1 - 5 - 2)
                 =  ((-8) >= 0 ? -8 : 1 - 7)
                 =  -6
```

Q4. Le fichier `module.c` contient l'implémentation du module. Cette implémentation peut introduire des déclarations de type, des déclarations de constante ou des macros-fonctions sans problème, qui peuvent être utilisées dans la partie privée du module. En revanche, il n'y a pas de raison d'ajouter des balises `#ifndef MODULE_H`/`#define MODULE_H`/`#endif` pour protéger contre les inclusions multiples dans le fichier `module.c`: c'est dans le fichier `module.h` que cette inclusion doit se trouver, justement pour empêcher l'inclusion multiple du fichier d'en-tête `module.h`.

Q5. Tout d'abord, on remarque que si la matrice n'est pas carrée, c'est-à-dire que son nombre de lignes est différent de son nombre de colonnes, alors elle est forcément non symétrique. Ensuite, on remarque qu'une matrice est symétrique si `m->valeurs[i][j] == m->valeurs[j][i]` pour tous indices valides `i` et `j`. Finalement, pour être plus efficace, on peut faire la vérification seulement dans les cas où `i < j`. Cela nous donne la fonction suivante:

```
bool is_symmetric(const struct Matrix* m) {
  if (m->r != m->c)
    return false;
  for (int i = 0; i < m->r; ++i)
    for (int j = i + 1; j < m->c; ++j)
      if (m->values[i][j] != m->values[j][i])
        return false;
  return true;
}
```